Computers & Fluids 93 (2014) 153–163

Contents lists available at ScienceDirect

Computers & Fluids

journal homepage: www.elsevier.com/locate/compfluid

On the implementation of low-dissipative Runge–Kutta projection methods for time dependent flows using OpenFOAM®

V. Vuorinen [a,*], J.-P. Keskinen [a], C. Duwig [b], B.J. Boersma [c]

[a] Department of Energy Technology, Aalto University School of Engineering, Finland
[b] Department of Energy Sciences, Lund University, Sweden
[c] Department of Energy Technology, Delft University of Technology, The Netherlands

ARTICLE INFO

Article history:
Received 20 September 2013
Received in revised form 11 January 2014
Accepted 23 January 2014
Available online 4 February 2014

Keywords:
OpenFOAM®
Runge–Kutta
Projection methods
Incompressible flows

ABSTRACT

Open source computational fluid dynamics (CFD) codes provide suitable environments for implementation, testing and rapid dissemination of algorithms typically used for large-eddy simulations (LES) and direct numerical simulations (DNS). In particular, it is important to test low-dissipative algorithms in unstructured codes of industrial relevance. In the present paper, the implementation of incompressible, explicit Runge–Kutta (RK) based projection methods into the OpenFOAM® library is discussed. We search for low-dissipative alternatives to the second order time integration methods which are commonly used together with the standard pressure correction approach PISO (Pressure Implicit with Splitting of Operators) in many commercial and open source codes including OpenFOAM®. The practical implementation of the projection methods in OpenFOAM® is provided together with theory. The method is tested with the classical fourth order RK-method and the accelerated third order RK-method. Four numerical experiments are carried out in order to cross-validate the solvers and in order to investigate the drawbacks/benefits of the solution approaches. The test problems are: (1) 2d lid driven cavity flow at $Re = 2500$, (2) DNS of 3d turbulent channel flow at $Re_\tau = 180$, (3) LES of a 3d mixing layer, and (4) the 2d inviscid Taylor–Green vortex. The RK-methods are benchmarked against the standard OpenFOAM® LES/DNS solver based on the PISO pressure correction method. The results indicate that the RK projection methods provide low-dissipative alternatives to the PISO method. The turbulent test cases show also that the RK-methods have a good computational efficiency.

1. Introduction

During the past decades the numerical methods and simulation strategies for incompressible flows have become well established [1–7] and the methods have been applied in a wide range of applications [8–21]. Ideally, as suggested by academic studies, LES and DNS would not only require computational power, but also relatively accurate numerics [1–8]. In particular, high-order time-integration and spatial discretization methods would be preferred for ensuring minimal influence of numerical diffusion and dispersion on the flow. Additionally, the chosen approach for pressure–velocity coupling has to be carefully chosen for minimizing numerical errors [2–4]. While high order methods are common in academic LES/DNS studies [1,5–8], low order methods are still, in the standard implementation of the codes, typically applied for LES in commercial codes such as Fluent [9] or Star-CD [10,11] and also in open source codes such as Kiva [12,13] and OpenFOAM® [14–16,20–31]. In order to carry out high-resolution LES in industrial applications, low-dissipative methods need to become commonly available also in commercial and open source codes. Descriptions on how to practically implement the algorithms into open source codes are scarce in the literature and thereby further assessed in this paper.

The present study deals with the practical implementation and testing of existing, well established numerical methods suitable for LES/DNS in an open source CFD code. As a test platform, we choose the unstructured, finite volume method (FVM) based open source C++ library OpenFOAM®. OpenFOAM®, as released under the Gnu Public License (GPL), has gained a vast popularity during the recent years [14–16,20–31]. The readily existing solvers and tutorials provide a quick start to using the code. The existing solvers can also be freely modified in order to create new solvers regarding existing [21–24] and new solution [25,29,31] approaches. Here, we depict one of the outcomes of a three year project that aimed at investigating how low-dissipative pressure–velocity coupling approaches

* Corresponding author. Tel.: +358 405579376.
E-mail address: ville.vuorinen@aalto.fi (V. Vuorinen).

http://dx.doi.org/10.1016/j.compfluid.2014.01.026
0045-7930/

and high order time integration methods can be implemented into OpenFOAM®.

Various methods for coupling velocity and pressure in incompressible flows have been presented in the literature. In the present paper the main interest is on the projection method [2] which we compare against the standard PISO solver in the OpenFOAM® code [3]. PISO aims at coupling pressure and velocity in simulations that use implicit time integration [3,4]. The operator splitting allows for fast solution during each timestep while the use of implicit time integration allows larger timesteps than allowed by explicit time integration methods. The PISO algorithm consists of one predictor step, where an intermediate velocity field is solved using pressure from the previous timestep, and of a number of corrector steps, where intermediate and final velocity and pressure fields are obtained iteratively. The number of used corrector steps affects the solution accuracy and usually at least two steps are used. An alternative to PISO is the projection method [1,2,5,8,29,32] which was first introduced in the 1960s by Chorin [33] and Temam [34]. In contrast to PISO, a projection method does not involve a corrector loop but the velocity field is projected onto its solenoidal counter part using the pressure gradient in a single projection step. In RK-methods the projection is carried out in-between the integration substeps in order to retain the time accuracy [1]. Thereby, the projection method is simpler to implement than PISO. The projection method is usually applied with explicit time integration methods, such as the Runge–Kutta methods [1,2,5] and the method has been previously applied also in finite volume studies [29,35].

A common problem in incompressible flows on colocated grids is the possible decoupling of velocity and pressure. The velocity and pressure may become decoupled if the discretization is implemented inconsistently, see e.g. [2]. The presence of the spurious pressure modes (the odd–even decoupling or the checkerboard effect) depends on the discretization of the equations [3]. The basic strategies for avoiding the odd–even decoupling are (1) to use the Rhie–Chow interpolation [36] or (2) to apply a staggered grid instead of a colocated grid [2,5]. In the existing LES/DNS solvers of OpenFOAM® the pressure–velocity decoupling is avoided by resorting to a variant of the Rhie–Chow interpolation. This yields a non-constant matrix for the Poisson equation [36]. However, the use of Rhie–Chow interpolation is unnecessary even on colocated grids when projection methods with explicit time integration methods are employed [1,2]. Rather recently, Hirsch [2] has presented a straightforward implementation of projection methods on colocated, unstructured grids, where a constant matrix appears in the discretization of the Poisson equation.

Runge–Kutta methods in conjunction with FVM are very well known [35] and they have been widely applied especially in academic studies involving LES and DNS [1,5,6,8,29,30]. Also, new developments on the accelerated RK-methods, aiming at code speed-up, exist in the literature [37]. The implementation of RK-projection methods into OpenFOAM® is interesting for three reasons. First, OpenFOAM® relies on the second order implicit time integration so increasing the accuracy to third or fourth order has potential advantages [37]. It is also important to understand the potential speed-up gain when an accelerated RK-method is employed in contrast to the standard RK4-method [37]. Second, it is of interest to see how the projection method performs in comparison to the standard PISO method. Third, the projection method avoids the corrector loop which significantly simplifies the code implementation making the algorithm easier to implement.

The objectives of the paper are to (1) present how Runge–Kutta projection methods can be implemented into OpenFOAM® and (2) benchmark the Runge–Kutta methods against the standard PISO method of OpenFOAM®.

2. Governing equations and numerical approach

2.1. Incompressible Navier–Stokes equations

In the present paper we implement flow solvers for incompressible flows as described by the Navier–Stokes equations:

$$\frac{\partial \mathbf{u}}{\partial t} + \nabla \cdot (\mathbf{uu}) = -\nabla p + \nu \Delta \mathbf{u}, \tag{1}$$

where the convective term is $\mathbf{C} = \nabla \cdot (\mathbf{uu})$, the pressure gradient is $\mathbf{P} = -\nabla p$, and the viscous term is $\mathbf{D} = \nu \Delta \mathbf{u}$. Incompressibility of the fluid is guaranteed by the continuity equation

$$\nabla \cdot \mathbf{u} = 0. \tag{2}$$

Taking divergence from both sides of Eq. (1) leads to the Poisson equation for pressure

$$\Delta p = -\nabla \cdot \mathbf{C}. \tag{3}$$

Thereby, pressure and velocity are coupled in an elliptic manner requiring numerical solution of an algebraic group of equations.

2.2. Spatial discretization

2.2.1. Discretization of the convection term

In the finite volume method (FVM) the governing partial differential equations are integrated over a computational cell, after which the Gauss theorem is applied to convert the volume integrals into surface integrals involving surface fluxes [2]. The FVM in OpenFOAM® has been previously described in the literature for problems involving linear elasticity and fluid dynamics [22–24]. For example, in the momentum equation the convection term can be discretized with the FVM as follows:

$$\begin{aligned} \mathbf{C}_{fvm} &= \frac{1}{\Omega}\int_{\Omega} \nabla \cdot (\mathbf{uu}) d\Omega = \frac{1}{\Omega}\int_{\partial\Omega} \mathbf{uu} \cdot \mathbf{n}\, dS \\ &\approx \frac{1}{\Omega}\sum_{f=1}^{N} \mathbf{u}_f \mathbf{u}_f^c \cdot \mathbf{n}_f\, dS_f, \end{aligned} \tag{4}$$

where $\mathbf{n}_f$ is the outer normal of cell face f, the summation covers the faces in a given cell and Ω is the cell volume. In the present study, and also typically in OpenFOAM® solvers, the convective flux $\mathbf{u}_f^c \cdot \mathbf{n}_f\, dS_f$ is always interpolated to the cell faces linearly from the adjacent cells. Also the convecting variable $\mathbf{u}$ needs to be approximated on the cell faces to get the face value $\mathbf{u}_f$. For this purpose any available interpolation method, such as linear or upwind interpolation, TVD-scheme, or a recently developed scale-selective discretization (SSD) [29] can be applied.

2.2.2. LES based on scale-selective discretization

In implicit LES (ILES) of high Reynolds number flow the diffusive numerical discretization of the convective terms acts as a standalone turbulence model [7]. The SSD scheme offers a strategy to apply numerically robust, upwind biased discretization only to the fluctuating part of the convecting quantity. The SSD scheme has been analyzed in detail by Vuorinen et al. [29] and applied previously in LES of turbulent channel flow at $Re_\tau = 590$ [29] and in LES of highly compressible, supersonic jets at $Re \sim 10^5$ [30].

Next, we briefly outline the main features of the SSD scheme for FVM. The SSD scheme involves a pre-processing step where a high-pass (hp) filtered velocity $\mathbf{u}'$ is evaluated using a Laplacian filter. Then, the velocity field is written as $\mathbf{u} = \mathbf{u} - \mathbf{u}' + \mathbf{u}'$ from which $\mathbf{u} - \mathbf{u}'$ is identified as the smooth, low-pass filtered velocity and $\mathbf{u}'$ is the non-smooth velocity. Based on our numerical tests, we apply the scale separation procedure only for the convecting velocity in the convection term whereas the other terms in the

Navier–Stokes equation are left intact. Thereby, the SSD form of Eq. (4) becomes

$$\mathbf{C}_{ssd} = \frac{1}{\Omega}\int_{\Omega} \nabla\cdot[(\mathbf{u}-\mathbf{u}'+\mathbf{u}')\mathbf{u}]d\Omega = \frac{1}{\Omega}\int_{\partial\Omega}(\mathbf{u}-\mathbf{u}'+\mathbf{u}')\mathbf{u}\cdot\mathbf{n}\,dS. \tag{5}$$

The discrete approximation of Eq. (5) is

$$\mathbf{C}_{ssd} \approx \frac{1}{\Omega}\sum_{f=1}^{N}(\mathbf{u}_f-\mathbf{u}'_f+\mathbf{u}'_f)\mathbf{u}^c_f\cdot\mathbf{n}_f\,dS_f = \mathbf{C}^1_{ssd}+\mathbf{C}^2_{ssd}, \tag{6}$$

where

$$\mathbf{C}^1_{ssd} = \frac{1}{\Omega}\sum_{f=1}^{N}(\mathbf{u}_f-\mathbf{u}'_f)\mathbf{u}^c_f\cdot\mathbf{n}_f\,dS_f \tag{7}$$

contains the smooth term $\mathbf{u}_f-\mathbf{u}'_f$ which is discretized using the linear interpolation and the remaining part

$$\mathbf{C}^2_{ssd} = \frac{1}{\Omega}\sum_{f=1}^{N}\mathbf{u}'_f\mathbf{u}^c_f\cdot\mathbf{n}_f\,dS_f \tag{8}$$

contains the non-smooth part which is interpolated using a blend of upwind and centered scheme. Similar to the previous studies [29,30], we choose the blending factor for the centered part as $\alpha = 0.3$ resulting in a relatively strong upwinding by factor $1-\alpha = 0.7$ for $\mathbf{u}'_f$. The key point of the SSD scheme is that the upwind-biased scheme is only applied to the hp-filtered field component leading to low level of numerical diffusion. Formal analysis reveals that the overall spatial accuracy of the SSD scheme is of second order on uniform grids [29]. The diffusive truncation error of the SSD scheme acts as a third order damping term comparable to previous higher order implicit LES [39].

2.3. Temporal discretization using Runge–Kutta methods

Explicit Runge–Kutta methods are standard methods for simulating problems of form $dy/dt = f(t,y)$. RK-methods have been widely used in incompressible [1,2,5] and compressible [6,29,30] fluid simulations. In the present study we use as a base line method the classical fourth order RK-method (RK4) in which the solution y_n progresses to y_{n+1} according to the well known update sequence [1,2,38]

$$\begin{cases} k_1 = f(t_n,y_n)\Delta t,\\ k_2 = f(t_n+\Delta t/2, y_n+k_1/2)\Delta t,\\ k_3 = f(t_n+\Delta t/2, y_n+k_2/2)\Delta t,\\ k_4 = f(t_n+\Delta t, y_n+k_3)\Delta t,\\ y_{n+1} = y_n+(k_1+2k_2+2k_3+k_4)/6,\\ t_{n+1} = t_n+\Delta t. \end{cases} \tag{9}$$

The accelerated Runge–Kutta (ARK) methods [37] are targeted to providing code speed-up and they are based on using the information from two different timesteps. Thereby, the new value y_{n+1} is computed using both y_n and y_{n-1} instead of using only y_n as in the ordinary Runge–Kutta methods. Several variants of the ARK-methods have been given by Udwadia et al. [37]. Here, we use a third order ARK-method which can be written as follows:

$$\begin{cases} k_1 = f(t_n,y_n)\Delta t,\\ k_{-1} = f(t_{n-1},y_{n-1})\Delta t,\\ k_2 = f(t_n+5\Delta t/12, y_n+5k_1/12)\Delta t,\\ k_{-2} = f(t_{n-1}+5\Delta t/12, y_{n-1}+5k_{-1}/12)\Delta t,\\ y_{n+1} = y_n+(k_1+k_{-1}+2k_2-2k_{-2})/2,\\ t_{n+1} = t_n+\Delta t. \end{cases} \tag{10}$$

As can be noted, k_{-1} at the timestep n is same as k_1 of timestep $n-1$. Hence by storing the computed values of k_1 and k_2 on each time step, one can avoid the two evaluations of the function f on each time step. This allows the ARK3 to achieve higher efficiency than the respective ordinary RK3-method. However, as information from two previous timesteps is needed, the first timestep needs to be computed using a method which does not require such knowledge. For ARK3 simulations, we use the classical RK4-method for the first timesteps and continue with ARK3 after gathering the necessary history information. For the RK4 method four projections per timestep is required whereas only two projections per timestep is needed for the ARK3 method. For the conventional (non-accelerated) RK3 method the number of projections would be three per timestep. In fact, relatively trivial code speedup can be expected when the number of projection step is reduced. Here, we limit the studies to RK4 and ARK3 and discuss the anticipated speed-up gain together with comparison to PISO method later on in the paper.

2.4. Projection method

2.4.1. Helmholtz–Hodge decomposition

The Helmholtz–Hodge (HH) decomposition forms the mathematical basis for coupling velocity and pressure in incompressible flows. In fluid dynamics, the HH-decomposition states that any velocity field $\mathbf{u}$ may be expressed as a sum two parts: a divergence free and a curl free part [1]. In the HH-decomposition, a pressure p is required such that:

$$\mathbf{u} = \mathbf{u}^* - \nabla p. \tag{11}$$

In Eq. (11) $\nabla\cdot\mathbf{u}^* \neq 0$, $\nabla\cdot\mathbf{u} = 0$ and $\nabla\times\nabla p = 0$ since the gradient of any scalar field is curl-free. Taking divergence of Eq. (11) shows that p must be a solution of the Poisson equation

$$\Delta p = \nabla\cdot\mathbf{u}^*. \tag{12}$$

2.4.2. Projection method in colocated grids

Recently, Hirsch has shown how the projection method can be implemented in colocated, unstructured grids where all the flow variables are stored in the cell centers [2]. Importantly, the given approach does not require Rhie–Chow type interpolation which can dissipate kinetic energy of the flow [41]. In the following, the main aspects of the algorithm are explained for the Euler method as the Runge–Kutta method contains a sequence of similar update stages. First, an intermediate velocity $\mathbf{u}^*$ is evaluated using only the convective ($\mathbf{C}$) and the diffusive ($\mathbf{D}$) terms. Applying the Euler method over a timestep Δt, $\mathbf{u}^*$ can be expressed as

$$\mathbf{u}^* = \mathbf{u}^n - \Delta t(\mathbf{C}-\mathbf{D}), \tag{13}$$

which, in general, is not divergence-free. After this, the resulting field is projected back to its divergence-free component using Eq. (11) when p has been solved from Eq. (12).

To solve p, the Laplacian operator Δ needs to be discretized properly to avoid the decoupling of velocity and pressure. In the standard FVM Eq. (12) is solved by writing $\Delta p = \nabla\cdot\nabla p$ which is first integrated over a control volume Ω as follows

$$\frac{1}{\Omega}\int_{\Omega}\nabla\cdot\nabla p\,d\Omega = \frac{1}{\Omega}\int_{\Omega}\nabla\cdot\mathbf{u}^*\,d\Omega. \tag{14}$$

Using the Gauss' theorem, one obtains

$$\frac{1}{\Omega}\int_{\partial\Omega}\nabla p\cdot\mathbf{n}\,dS = \frac{1}{\Omega}\int_{\partial\Omega}\mathbf{u}^*\cdot\mathbf{n}\,dS, \tag{15}$$

where $\int_{\partial\Omega}$ denotes an integral over the boundary faces of the control volume Ω. The key point of the Hirsch algorithm [2] is that in Eq. (15) the boundary integral involves ∇p defined at the cell faces. In order to emphasize this, we write

$$\frac{1}{\Omega}\int_{\partial\Omega}(\nabla p)_f\cdot\mathbf{n}\,dS=\frac{1}{\Omega}\int_{\partial\Omega}\mathbf{u}_f^*\cdot\mathbf{n}\,dS. \tag{16}$$

Eq. (16) can be discretized as follows

$$\frac{1}{\Omega}\sum_{f=1}^{N}(\nabla p)_f\cdot\mathbf{n}_f\,dS_f=\frac{1}{\Omega}\sum_{f=1}^{N}\mathbf{u}_f^*\cdot\mathbf{n}_f\,dS_f, \tag{17}$$

where $\mathbf{n}_f$ is the outer normal of the cell face f, dS_f is the area of the cell face f, the summation covers the N faces in a given cell and Ω is the cell volume. The formation of a non-compact stencil can be avoided by evaluating $(\nabla p)_f$ at the cell face directly in contrast to first evaluating the gradients at the cell centroids with subsequent interpolation to the cell face. For example, the cell face gradient operator between the cells i and $i+1$ would only use the values p_i and p_{i+1}. In an unstructured, second order accurate code, this leads to a compact stencil for the Laplacian operator when the terms in Eq. (17) are collected together into a matrix equation. In 3d, uniform Cartesian grid, Eq. (17) is equivalent to the standard seven-point stencil i.e.

$$\Delta p\approx\frac{p_{i+1,j,k}+p_{i-1,j,k}+p_{i,j+1,k}+p_{i,j-1,k}+p_{i,j,k+1}+p_{i,j,k-1}-6p_{i,j,k}}{\Delta x^2}. \tag{18}$$

As mentioned by Hirsch, this approach is colocated as the variables are stored in the cell centers. Yet, in a way the algorithm applies a staggered evaluation of the cell face pressure gradient [2]. Thereby, the shown approach avoids the $(i-2,i,i+2)$ stencil in the discretization of the Laplacian operator and the formation of the spurious pressure modes can be avoided [2,41]. In OpenFOAM® the Laplacian operator is discretized on a compact stencil readily.

2.5. Numerical approach for the Poisson equation

The solution of the Poisson equation is the major time consumer per iteration and, thereby, any acceleration strategy results typically in a substantial speed-up gain. The elliptic nature of the problem advocates a multigrid solution. Here we use an agglomerative algebraic multigrid (AAMG) procedure with a Gauss–Seidel linear solver as preconditioning followed by a conjugate gradient procedure. The AAMG uses a V-cycle with 2 post-sweeps for decreasing the residual by a factor 20. Furthermore, a conjugate gradient solver is used to decrease the residual further down to a relative tolerance of 10^{-7}. This strategy has been found successful in the past [40] for solving efficiently the pressure–velocity coupling, supporting thereby the present choice.

2.6. A note on PISO implementation in OpenFOAM®

It is not guaranteed that the implementation of the solvers in any CFD code, including OpenFOAM®, is faithful to the original theory behind the algorithms. As in all CFD-codes, the implementation can be designed to involve stabilization terms resulting in robust simulations also on low quality meshes. This may increase the numerical dissipation in the solution. In commercial solvers such stabilization terms cannot be tracked but in open source codes the code is free for evaluation. In the following we briefly discuss one detail of the PISO method implementation that can cause artificial dissipation and is not part of the original algorithm [3].

A further look at the PISO solver code reveals that in the pressure correction loop the mass flux on timestep n is of the form $\phi^n_{PISO}=\phi_1+\phi_2$. The first term is the standard mass flux $\phi_1=\mathbf{u}^n\cdot\mathbf{n}\,dS$ that we use also in the RK-solvers ('flux without extra term'). The latter term ϕ_2 is an undocumented correction term which is specific to OpenFOAM® ('extra term') and it is not a part of the original PISO implementation [3]. In the present PISO investigations we use the OpenFOAM® default formulation with the correction term. The flux ϕ_2 is evaluated on each cell face as follows

$$\phi_2=\frac{\gamma}{\Delta t}\left(\left[\frac{1}{A_p^n}\right]_f\phi^{n-1}_{PISO}-\left[\frac{1}{A_p^n}\mathbf{u}^{n-1}\right]_f\cdot\mathbf{n}\,dS\right), \tag{19}$$

where $\mathbf{u}^{n-1}$ is the velocity from the previous timestep, and A_p^n is the matrix diagonal coefficient of the implicitly discretized momentum equation [3]. The brackets $[\cdot]_f$ denote an operator for evaluating first the quantity inside the brackets at cell centers followed by linear interpolation to the cell face f, and the coefficient $0\leqslant\gamma\leqslant1$ is evaluated as follows

$$\gamma=1-\min\left(\frac{|\phi^{n-1}_{PISO}-\mathbf{u}^{n-1}\cdot\mathbf{n}\,dS|}{|\phi^{n-1}_{PISO}|+\epsilon},1\right), \tag{20}$$

where ϵ is a very small number to avoid division by zero. It is seen that the overall effect of ϕ_2 is to influence the divergence of the face velocity since the interpolated velocity becomes subtracted from the flux. Since A_p depends on time and velocity, Eq. (19) is non-linear. It can be argued that the difference of two non-equal face interpolants may lead to a diffusive flux. Thereby, a diffusive term (similar to a first order damping term) may be introduced to the flux via ϕ_2. The influence of the diffusive properties of ϕ_2 is further studied in Section 4.4.

3. Implementation using OpenFOAM®

3.1. High-level scripting and case setup

The OpenFOAM® package includes several examples on writing high-level solvers for various applications. In order to implement a new solver for the projection method we choose as a starting point an existing incompressible solver and delete the script inside the main `while`-loop. The implementation of RK-methods involves then straighforward operations that are carried out using the velocity, pressure and flux fields as is explained in the following sections. The case for a new solver is set in OpenFOAM® always in the same manner following the examples given in the various tutorials: boundary conditions, mesh, solver settings and discretization settings are given in a standard folder structure.

3.2. Data types for fields and operators

The previous works [22–26] have introduced several aspects of OpenFOAM® programming. Here a brief summary is given for completeness. The main data types needed in the simulation of PDE's in cell-centered, unstructured, colocated grids are scalar, vector and tensor fields defined at the cell centroids. For example, to define a velocity field at cell centers one uses the `volVectorField` data type. Table 2 summarizes some of the other available types in brief. In the FVM one also needs to solve for fluxes on the cell faces. Hence, additional data types, termed `surfaceScalarField` and `surfaceVectorField` as defined at the cell face centers, are also needed. Also interpolation methods need to be implemented in order to interpolate the cell centered field values to the cell faces. Functionality for evaluating partial differential operators such as $\nabla,\nabla\times$, and Δ is required as well. Among other data types, these data types and differential operators are provided in the OpenFOAM® toolbox thereby offering efficient Field Operation and Manipulation (FOAM) [26]. A brief list of the differential operators is given in Table 1 and an overview of some of the most important data types is given in Table 2.

Table 1
Examples of differential operator types in OpenFOAM®. The `fvc` syntax declares the operators to be explicit i.e. to be evaluated with known values [26].

Operator	Syntax	Output for a scalar	Output for a vector
∇	`fvc::grad(F)`	`volVectorField`	`volTensorField`
$\nabla\times$	`fvc::curl(F)`	–	`volVectorField`
Δ	`fvc::laplacian(F)`	`volScalarField`	`volVectorField`

Table 2
Examples of field data types in OpenFOAM® [26].

Field name	Variable	Data type	Variable values defined at
Velocity	`U`	`volVectorField`	Cell center
Pressure	`p`	`volScalarField`	Cell center
Rate of strain	`S`	`volTensorField`	Cell center
Velocity flux	`phi`	`surfaceScalarField`	Cell face center

3.3. Implementation of boundary conditions

In RK solvers the boundary conditions need to be consistently updated in between the RK substeps. The way to do this is to manually 'hard code' the update stages of the boundary conditions in the high-level flow solver. For example, for the velocity field, this can be conveniently carried out by applying the syntax.

```
U.correctBoundaryConditions() ;
```

which updates the boundary conditions as defined in the input files. Similarly, boundary field values can be explicitly manipulated with the syntax

```
U.boundaryField() .
```

3.4. Implementation of the projection method using OpenFOAM®

In OpenFOAM®, the projection method can be described the easiest with the explicit Euler method i.e. the original Chorin–Temam algorithm [1,2].

Step 0: Find the flux of convecting velocity (which is free of divergence) on the cell faces by taking the dot product of linearly interpolated velocity with cell face outer normal vector:

```
phi = interpolate(U)&mesh.Sf();
```

Step 1: Find the change in velocity at cell centers using the convection and the diffusion terms without the pressure gradient.

```
dU = dt * (-fvc :: div(phi, U) + nu * fvc :
    : Laplacian(U)).
```

Step 2: Update the velocity first at cell centroids only and then update the velocity boundary conditions

```
U = U + dU;

U.correctBoundaryConditions().
```

Step 3: Find the flux of velocity on the cell faces by taking the dot product of linearly interpolated velocity with cell face outer normal vector:

```
phi = interpolate(U)&mesh.Sf().
```

Step 4: Solve the Poisson equation for the pressure:

```
fvm :: Laplacian(p) = fvc :: div(phi).
```

Step 5: Project the velocity onto its divergence free component, first only at cell centers, and after that correct the boundary conditions:

```
U = U - fvc :: grad(p);

U.correctBoundaryConditions().
```

In the classical RK4 algorithm the above mentioned steps are just repeated four times. Thereby, also the Poisson equation needs to be solved four times per timestep. In our version we implemented the compact low storage version of RK4 (see e.g. [1]) so that during the four substeps the solution becomes eventually accumulated into a field variable. In order to reduce the number of projections per timestep from four to only two, and in order to speed up the computations, we also implemented the accelerated RK3-method (ARK3) which requires extra storage of δ-fields from the previous timesteps [37]. We note that, in contrast to the PISO method, in the present projection methods the Laplacian operator in Eq. (12) is always a constant matrix that only depends on the computational grid. Thereby, the Laplacian matrix only needs to be created once in the beginning of the simulation which makes the solution in *Step 4* more efficient.

3.5. Example: density based solver for the Euler equations

As a brief example, we demonstrate implementation of a density based RK4-solver for compressible flows using OpenFOAM®. In our previous studies we validated a similar solver on the lid-driven cavity flow and carried out LES of subsonic jets [31]. Later on, the same flow solver was modified to handle shock waves and LES of highly underexpanded jets was carried out using the developed solver [30]. In compressible flows, pressure p is directly linked to density ρ, temperature T and the gas constant R via the equation of state $p = \rho RT$ and the solution of Poisson equation is avoided. This is in contrast to the standard transient LES solvers of OpenFOAM® which are typically based on the compressible formulation of the PISO-method [3,4]. In order to promote the further development of density based compressible solvers into OpenFOAM®, we give a brief example of implementing the Navier–Stokes equations. For clarity, we only discuss the inviscid limit, the Euler equations, which read:

$$\frac{\partial \rho}{\partial t} + \frac{\partial \rho \mathbf{u}}{\partial x_j} = 0, \tag{21}$$

$$\frac{\partial \rho \mathbf{u}}{\partial t} + \nabla \cdot (\rho \mathbf{u}\mathbf{u}) = -\nabla p, \tag{22}$$

$$\frac{\partial \rho e}{\partial t} + \nabla \cdot ((\rho e + p)\mathbf{u}) = 0, \tag{23}$$

where $e = c_v T + \mathbf{u}^2/2$ and c_v is the specific heat of the gas. As in the incompressible case, the standard method of lines [38] requires the update of the solution at the cell centroids. Thereby, the δ-variables $\delta\rho$, $\delta\rho\mathbf{u}$ and $\delta\rho e$ need to be evaluated. An update scheme for a compressible flow may now be conveniently programmed as follows:

Step 0: Interpolate `rhoU` *and* `rho` *to the cell face using the* `interpolate` *function. The convection flux at the cell faces may then be found by dividing the interpolated values of* `rhoU` *and* `rho` *with each other and taking a dot product of the result with the cell face normal vector:*

```
phi = (interpolate(rhoU)/interpolate(rho))&mesh.Sf().
```

Step 1: Find the change of the conservative variables at the cell centers using the convection terms and the pressure gradient. Note that boundary condition update is unnecessary for $\delta\rho, \delta\rho\mathbf{u}$ *and* $\delta\rho e$.

```
drho = -dt * fvc :: div(phi, rho);
drhoU = -dt * (fvc :: div(phi, rhoU) + fvc :: grad(p));
drhoe = -dt * (fvc :: div(phi, rhoe + p)).
```

Step 2: In case of a one-step method the conservative variables could be readily updated with the computed increments.

```
rho = rho + drho;
rhoU = rhoU + drhoU;
rhoe = rhoe + drhoe.
```

Step 3: The primitive variables are updated at the cell centers:

```
U = rhoU/rho;
T = (rhoe - 0.5 * rho * magSqr(U))/Cv/rho;
p = rho * R * T.
```

Step 4: The boundary conditions need to be updated. For pressure and temperature the boundary conditions are read from input files whereas density is calculated based on the equation of state.

```
p.correctBoundaryConditions();
T.correctBoundaryConditions();
rho.boundaryField() = p.boundaryField()/
  (R * T.boundaryField()).
```

Modification of the outlined algorithm by viscous terms and shock capturing ability, as well as the extension to fourth order RK-method is straightforward using methods described in [30]. It should be noted that the syntax above works directly for 1d, 2d and 3d cases, arbitrary geometries, in parallel and for a wide variety of existing basic boundary conditions.

4. Numerical tests

Next, the implemented ARK3 and RK4 projection methods are tested on four prototype flows. Comparison with the standard PISO method of OpenFOAM® with second order backward time integration method is given. The aim is to (1) show that the implemented solvers work and give comparable results with one another and (2) compare the dissipation properties between the solvers.

4.1. The lid-driven cavity flow

The flow in a lid-driven cavity (LDC) is a well known benchmark problem of CFD-codes [2,42]. The setup consists of a 2D square box filled with incompressible fluid with viscosity ν. The box is enclosed by four walls of length L. The topmost wall is moving from left to right with velocity U_{lid} so that a vortex forms inside the cavity. Here we test the implemented two flow solvers at the Reynolds number of $Re = 2500$. The top part of Fig. 1 shows that on a relatively fine grid of 128^2 cells, all the solvers give a very close match with each other and agree very well with the reference data of Erturk et al. [42]. A grid convergence study for $N = 64^2,\ 128^2,\ 256^2$ and 512^2 grids at maximum Courant number $Co_{max} = 0.5$ is shown in the bottom part of Fig. 1 confirming the second order spatial accuracy of the solvers. The reference solution for each of the methods for the L2-error is the numerical solution at the finest grid level $N = 512^2$. The LDC-flow shows that (1) the developed solvers work and give almost indistinguishable results for a laminar flow and (2) the convergence behavior of the solvers is very similar. Next, more complicated examples are considered.

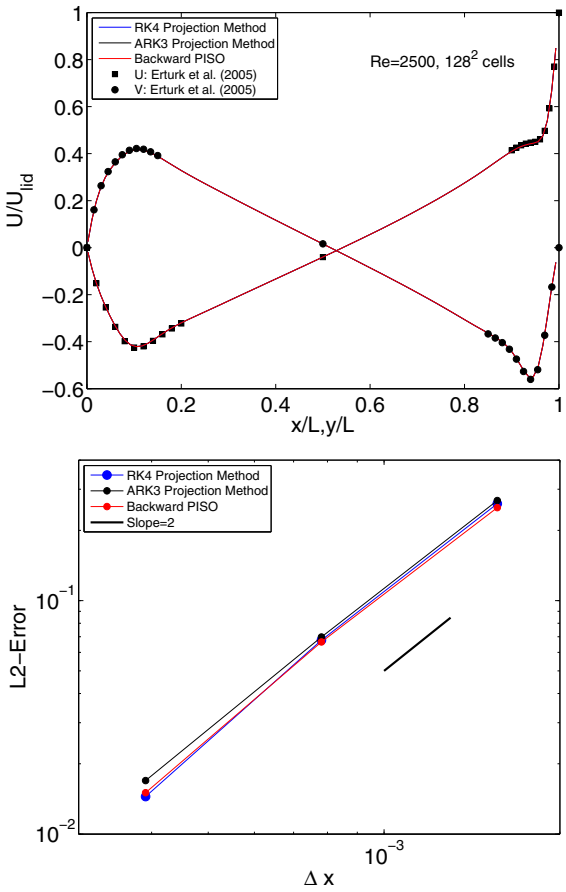

Fig. 1. Top: Velocity profiles along the horizontal and vertical centerlines. Bottom: L2-error for x-component of velocity.

4.2. DNS of turbulent channel flow at $Re_\tau = 180$

As an example of the capability of the implemented simulation codes we study the turbulent channel flow at a relatively low friction Reynolds number of $Re_\tau = 180$. The flow is driven by a body force i.e. a constant pressure gradient. The case is chosen as it is very well established and thereby it provides a good test problem for benchmarking the solvers. The size of the domain is $4\pi \times 2 \times 2\pi$ and contains 128^3 cells similar to the previous study by Moser et al. [8]. Fig. 2 shows plots of the mean streamwise velocity plotted on semilogarithmic scales. The velocities have been normalized with the friction velocity so that $u^+ = u/u_\tau$ and the viscous length scale so that $y^+ = yu_\tau/\nu$. Here we use the definition $u_\tau = \sqrt{\nu \frac{du}{dy}}$, where $\frac{du}{dy}$ is the wall normal velocity gradient evaluated numerically at the wall from the mean streamwise velocity. It is seen that all the solvers give a very good match with the DNS data by Moser et al. [8]. This is expected since the used computational grid is close to a DNS resolution. Fig. 2 also shows that the compressible, density based RK4-method, at Mach number $Ma \approx 0.3$ i.e. the incompressible limit, is able to capture the mean flow properly [30].

Fig. 2 shows also the diagonal Reynolds stresses. The RK-solvers give a very good match with each other and also with the standard PISO solver of OpenFOAM®. A slight mismatch is seen in the streamwise Reynolds stress for which all the studied solvers sligthly underpredict the data by Moser et al. [8] produced with spectral methods. The small deviation is understandable since second order spatial schemes, as used here, have poor dispersion properties in the high-wavenumber range. Fig. 3 shows that also the

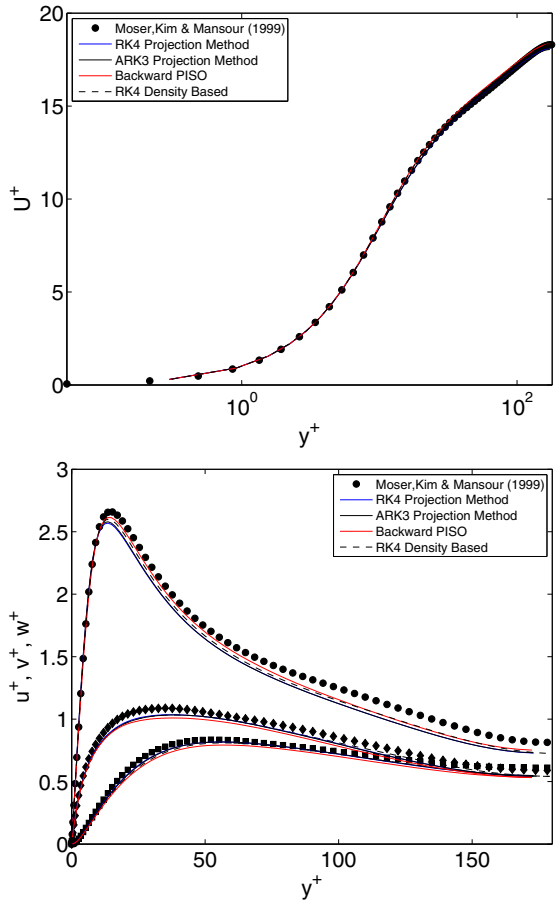

Fig. 2. Top: Mean streamwise velocity. Bottom: Diagonal components of Reynolds stress tensor.

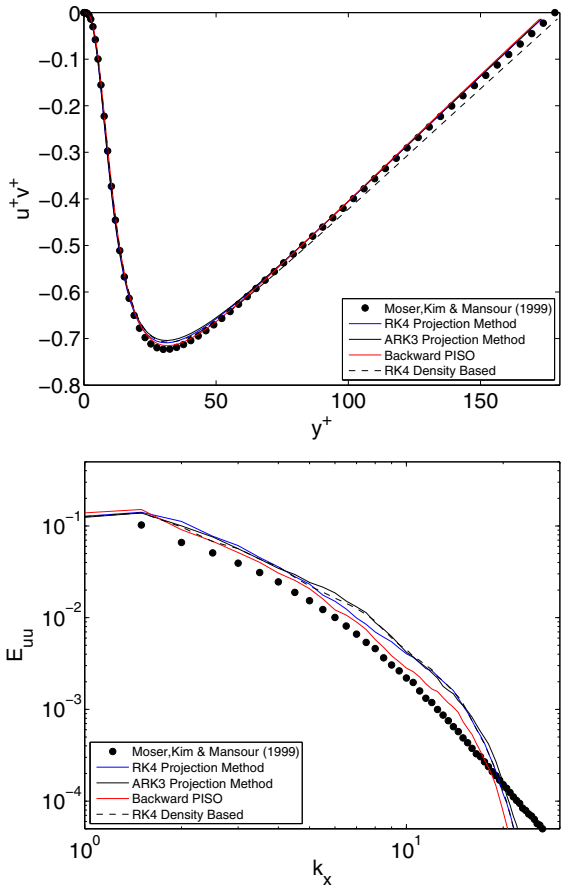

Fig. 3. Top: Off-diagonal component of Reynolds stress tensor. Bottom: Streamwise velocity spectrum.

off-diagonal stress term u^+v^+ is very well predicted by all the developed solvers. In general, for the Reynolds stresses, the solvers are very close to each other and it is hard to tell which of the approaches performs the best. However, the differences between present results and the DNS data become more clear when the spectrum is considered in Fig. 3. In particular, a fast decay around wavenumber $k_{cut} \approx \frac{k_{max}}{2} = \frac{\pi}{2\Delta x}$ is observed for the second order accurate finite volume methods.

4.3. Large-eddy simulation of 3d transitional shear layer

The two examples discussed above are flows where energy is constantly brought into the system. The simulations were also carried out on relatively fine resolutions so good results are not a surprise. Two questions are: (1) What if energy is not constantly brought to the system? (2) What if coarser resolutions are used in the presence of a dissipating LES model? Next, we discuss and carry out implicit LES based on scale-selective filtering [29] to answer these questions.

We consider the growth and transition of an instability in a 3d shear layer. The simulation domain is the box $0 \leqslant x \leqslant 4\pi,\ 0 \leqslant y \leqslant 2\pi$, and $0 \leqslant z \leqslant \pi$. The shear layer is initialized with the velocity field

$$\begin{cases} u(x,y,z,t=0) = U_o u(y), \\ v(x,y,z,t=0) = \frac{du}{dy}\sin(16x)A_o, \\ w(x,y,z,t=0) = \frac{du}{dy}\sin(16z)A_o, \end{cases} \tag{24}$$

where $u(y) = \tanh((y-\pi)/\sigma)$. The initial width, and thereby the level of shear, of the layer is determined by the parameter $\sigma = \pi/12$. The fixed perturbations in lateral directions are activated only in the region where shear occurs using the derivative $\frac{du}{dy}$. The amplitude of the perturbations is $A_o = 2$ whereas the initial velocity amplitude is $U_o = 50$. Before starting the simulation, divergence is removed from the initial field in a single projection step. Thereby, the final initial condition has a maximum perturbation level of 4% which is enough to trigger the instability reasonably fast. We note that this initial projection step is carried out also for the PISO method in order to ensure that the initial conditions between the solvers are exactly the same.

The numerical dissipation of the solution approaches is assessed by carrying out a grid resolution test on three different meshes with coarse ($128 \times 128 \times 64$ cells), intermediate ($192 \times 192 \times 96$ cells) and fine ($256 \times 256 \times 128$) resolutions. Similar to the previous study on a 2d shear layer [29] we evaluate the quality of the simulation by investigating the time evolution of the following integral properties: (1) the total kinetic energy $E_{kin}(t) = \int \frac{1}{2}\mathbf{u}^2\, dV$, (2) the total resolved viscous dissipation $\chi_{visc}(t) = \int \nu|\nabla\mathbf{u}|^2\, dV$, and (3) the total numerical dissipation $\chi_{num}(t)$. Numerical dissipation consists of the truncation errors along with the error associated with the pressure correction method. The total effect of these errors can be evaluated a posteriori due to the analytical connection where the total energy decay rate is dictated by the total contribution of χ_{visc} and χ_{num}:

$$\frac{dE_{kin}}{dt} = -\chi_{visc} - \chi_{num}. \tag{25}$$

Thereby, χ_{num} can be evaluated from the simulation data when $E_{kin}(t)$ and χ_{visc} are known.

Fig. 4 shows the Q-isosurface of the vortex roll-up and growth of the Kelvin–Helmholtz (KH) instability at the finest resolution. The forming KH-vortices are enveloped by rib vortices (RV's) which evolve rather slowly. The role of the RV's is to drive the growing 2d primary instability to a 3d turbulent state. It is noted that the details of the transition depend on the chosen method. Yet, all the methods predict the onset of the KH-instability to take place between $t = 0.5$–0.6 s.

Fig. 5 shows the decay of the kinetic energy and the resolved viscous dissipation. It is noted that (1) even on the finest resolution the rate of energy decay depends on the method to some extent, (2) for a given grid PISO and the projection methods all predict the onset of the instability, and (3) the RK-methods compare well with each other but differ mostly from the PISO method. It is also noted that, along with the gradients becoming better resolved, the fine grid resolves also more viscous dissipation than the intermediate grid. However, it is noted that at the intermediate and fine grids the location of the dissipation peaks is roughly the same for a given method. The height of the first peak (I) is approximately grid independent but the heights of the second peak (II) differ implying that even on the finest grid the results are still LES and not DNS.

It turns out that the projection methods resolve a larger amount of the viscous effects than the PISO method. The explanation is found in Fig. 6 where the evolution of χ_{num}/χ_{visc} is shown for the coarsest and the finest grids. It is noted that PISO involves higher levels of numerical dissipation than the projection methods. Numerical dissipation reduces gradients from the flow and thereby also reduces the resolved magnitude of χ_{visc}. At coarse resolutions, peak values for χ_{num}/χ_{visc} may be as high as 10–15 during the transition stage when large vortices merge together as seen also from Fig. 4. As noted in the previous study [29], when $\chi_{num}/\chi_{visc} \gg 1$ numerical dissipation dominates the molecular effects. However, at fine resolutions the gradients become better resolved and typically $\chi_{num}/\chi_{visc} < 5$. This is seen particularly well from the lower panel of Fig. 6 showing the time averages $\langle \chi_{num}/\chi_{visc} \rangle$ as a function of grid spacing. The numerical dissipation of PISO exceeds the dissipation of the RK-methods by 20–30%. This example indicates that the projection method dissipates less than the standard PISO method in OpenFOAM® and that this conclusion seems to be grid independent. We emphasize the fact that the reduced dissipation of the developed projection methods might result in some lack of robustness on bad quality meshes. In contrast, the noted higher dissipation of PISO may be an indication of better algorithmic robustness even on bad meshes.

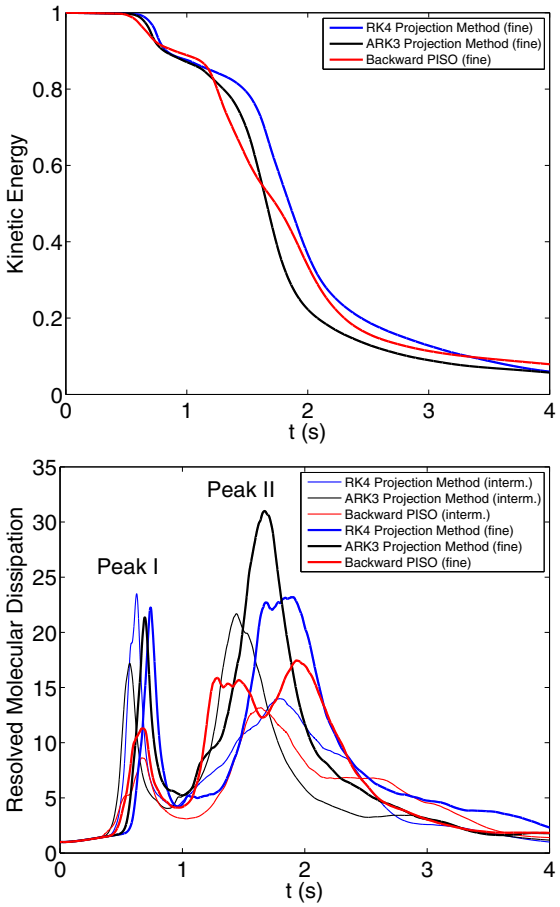

Fig. 5. Top: Total kinetic energy in a 3d temporally evolving shear layer the finest resolution. Bottom: Total resolved molecular dissipation in the shear layer for the intermediate and finest resolutions.

4.4. Laminar flow revisited: the inviscid Taylor–Green vortex

Above, it was observed that the standard PISO algorithm and the newly developed projection methods give comparable results with each other in laminar and turbulent flows at moderate Reynolds numbers. However, in most of the studied cases PISO showed

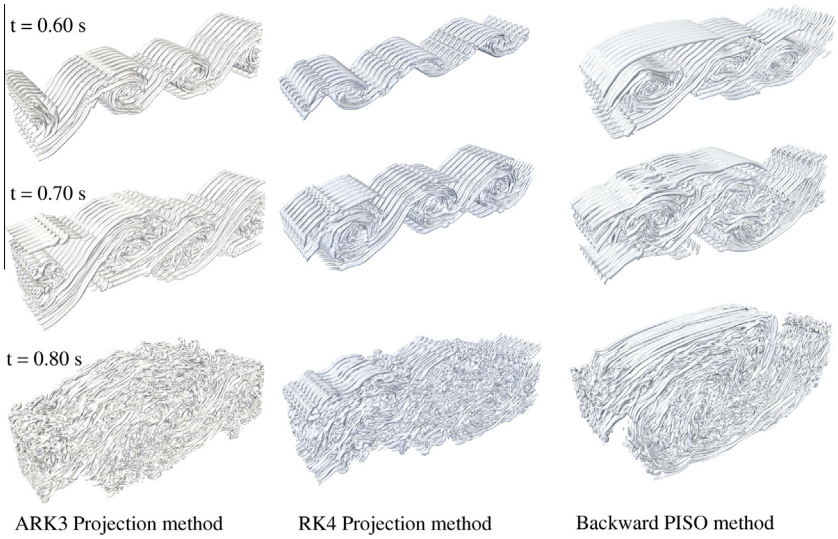

Fig. 4. The Q-isosurface $Q = 1000\ s^{-2}$. The temporal phase of the transition process depends on the method to some extent.

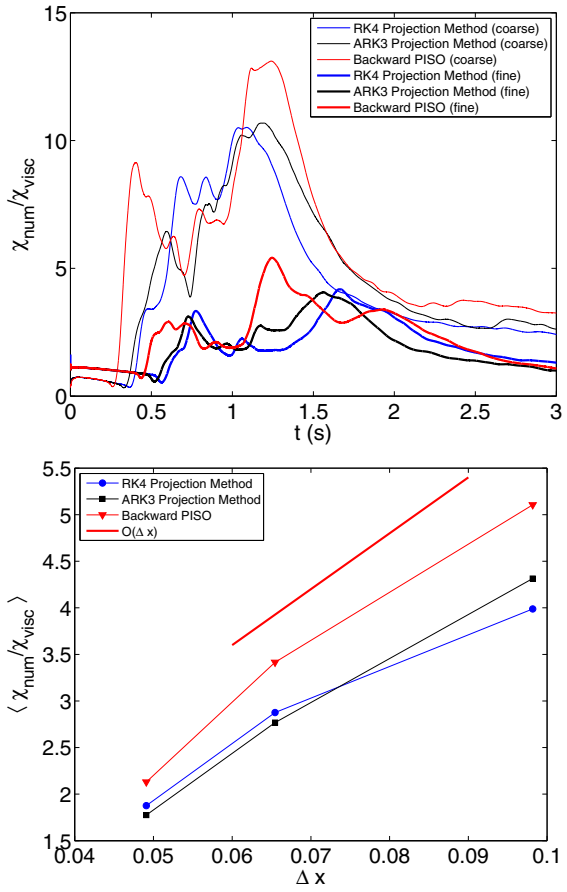

Fig. 6. Top: Ratio of numerical and viscous dissipation for the coarsest and the finest resolutions. Bottom: Time average of numerical and viscous dissipation vs grid spacing.

some differences in comparison to the projection methods. In particular, the results from the turbulent shear layer implied that the numerical dissipation of PISO on coarse grids was higher than for the projection methods in LES. The differences could originate from (1) the order of the time integration method, (2) the pressure treatment, and (3) the mass flux correction term ϕ_2 specific to the OpenFOAM® PISO implementation. A thorough timestep/grid comparison test on a simple, inviscid flow case can bring more insight to the origin of the errors.

As a final demonstration, we compare the implemented solvers with two versions of PISO: (1) 'flux without extra term' ($\phi_{PISO} = \phi_1$) and (2) 'flux with extra term' ($\phi_{PISO} = \phi_1 + \phi_2$). In the following, we investigate the sensitivity of results to timestep size and grid resolution in a very simple 2d setup. The 2d Taylor–Green (TG) vortex is a classical validation case for incompressible codes [1]. The problem concerns an array of periodic vortices described as

$$\begin{cases} u(x,y,t) = \sin(x)\cos(y)\exp(-\nu t), \\ v(x,y,t) = -\cos(x)\sin(y)\exp(-\nu t), \\ p(x,y,t) = \frac{1}{4}(\cos(2x)+\cos(2y))\exp(-2\nu t), \end{cases} \tag{26}$$

where $0 \leqslant x \leqslant 2\pi,\ 0 \leqslant y \leqslant 2\pi$. We study the inviscid limit $\nu = 0$ where the vortex remains unchanged. Thereby, the total kinetic energy of the vortex is conserved i.e. $dE_{kin}/dt = 0$. Fig. 7 shows that in practice $dE_{kin}/dt < 0$ due to numerical dissipation. It is obvious that within a few integral timescales $T_o = 2\pi s$ the standard PISO ('flux with the extra term') has dissipated a significant portion of kinetic energy. For example, when the maximum Courant number $Co_{max} = 0.5$, at time $t = 2T_o$ on the 32^2 grid the energy level has decreased by factor 0.8. Respectively, for the finest 128^2 grid the damping factor is 0.98 which is still relatively high. When $Co_{max} = 0.06125$ the dissipation is clearly reduced. We link this strong decay of energy to ϕ_2 because, when ϕ_2 is removed, PISO gives comparable results with the RK projection methods which are noted to be very non-dissipative. The error for the coarser grid is of order $\sim 10^{-3}$ and for the finest grid the error is only of order $\sim 10^{-5}$. Thereby, the kinetic energy is almost fully conserved for the RK projection methods in the inviscid limit. We note that the standard PISO-method improves significantly with the grid resolution and the PISO becomes comparable with the projection methods when simulations were carried out without the correction term.

The grid dependency and timestep dependency of the dissipation errors is studied in Figs. 8 and 9 respectively. The projection methods are noted to be at least an order of magnitude less dissipative than the standard PISO method. When the flux in the PISO method is evaluated in the same way as in the RK-methods, PISO becomes comparable with the ARK3-method. For all the investigated spatial and temporal resolutions, the RK4-method dissipates the least. The dissipation errors can be also reduced quite significantly by choosing a smaller timestep. The influence of timestep on diffusive/dispersive errors is usually completely ignored and thereby it might be important to consider this aspect in future studies if possible. In fact, the previous investigations by Sengupta et al. [43] and Rajpoot et al. [44] have already raised concern regarding the temporal errors and interaction between temporal and spatial schemes in LES/DNS.

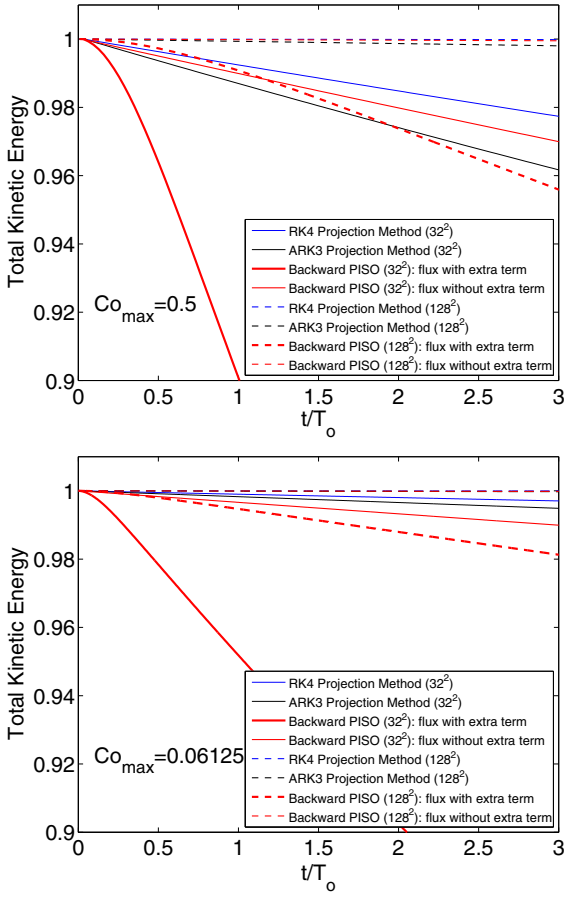

Fig. 7. Total kinetic energy in the inviscid 2d Taylor–Green vortex. The influence of timestep on dissipation errors is clear. In PISO the dissipation error is reduced by computing the flux without the robust formulation.

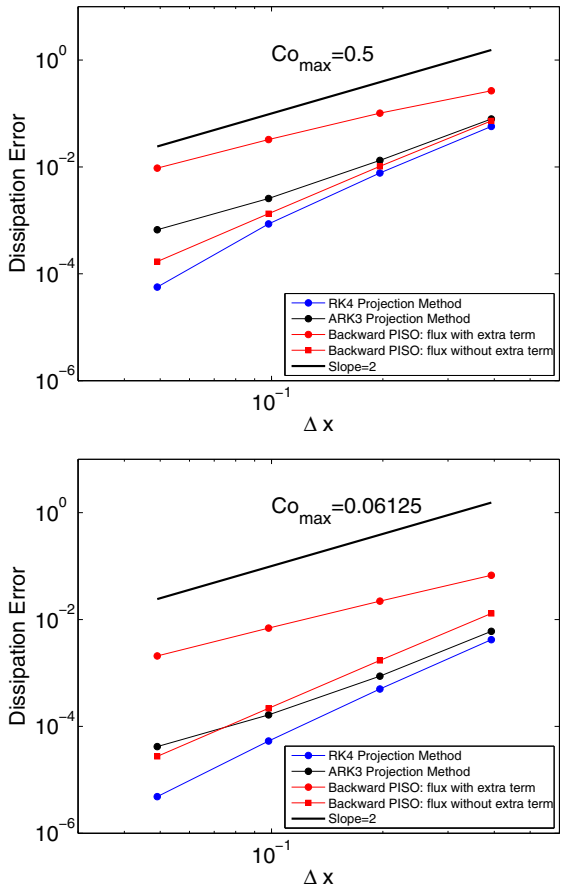

Fig. 8. Effect of grid resolution on the dissipation error in the inviscid Taylor–Green vortex at $Co_{max} = 0.5$ (top) and $Co_{max} = 0.06125$ (bottom).

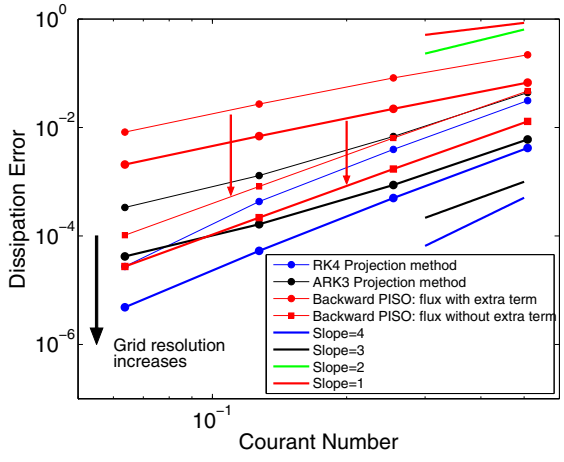

Fig. 9. Effect of timestep (Courant number) on the dissipation error. The thin line shows the error for 32^2 mesh and the thick line for 128^2 mesh. The red arrows show the improvement in results when a less robust flux computation in PISO is employed. (For interpretation of the references to color in this figure legend, the reader is referred to the web version of this article.)

4.5. Code execution times

The present simulations were carried out at two different parallel computing environments: Cray XT4/XT5 in Finland and IBM pSeries 575 in the Netherlands. Table 3 summarizes the code execution times for the turbulent flow cases i.e. the channel flow and the shear layer on the finest grid resolution. In all the runs the cell load per CPU is 65536. It is observed that on both systems the ARK3-method is 60–70% faster than PISO. RK4-method is slightly faster than PISO on the IBM system but nearly 50% faster than PISO on the Cray system. The speed-up can be explained by (1) explicit vs implicit time integration and (2) the differing approaches to treat the pressure. The two projection steps of ARK3 compared to the four projection steps of RK4 method explain why ARK3 is typically faster than the RK4 method by factor of the order $2/4 \sim 0.5$. We note that in general the CPU cost depends on the strategy to solve the Poisson equation, the load per CPU and the simulation case. Furthermore, for the PISO-method the number of corrector steps, which was here set to 2 as usual on orthogonal grids, has an influence on the total execution time. Thereby, this execution time comparison might depend on the solution strategy and the simulation case. However, the results show a noticeable speed-up by the RK projection methods.

Table 3
Computational time for the DNS and LES cases per timestep. Results are normalized with the respective execution time for the PISO solver.

Execution time	ARK3	RK4	PISO	CPU's	Cells/CPU	System
Channel flow	0.40	0.97	1	32	65536	IBM pSeries 575
Shear layer	0.30	0.53	1	128	65536	Cray XT4/XT5

5. Conclusions

In the present paper we showed how explicit Runge–Kutta based projection methods can be implemented into the OpenFOAM® code. To our knowledge, this paper is the first study to assess development and programming of RK-methods into OpenFOAM®. The findings have relevance also to commercial codes (e.g. CFX, Star-CD, Fluent) since RK projection methods are not available in such codes yet. The main findings of the paper are summarized as follows:

1. Runge–Kutta projection methods can be efficiently implemented with a few tens of lines using the high-level programming syntax and field algebra of the OpenFOAM® code.
2. Results from parallel runs for two turbulent flows showed computational speed-up of RK projection methods with respect to PISO. Simulations using ARK3-method were 60–70% faster than the respective PISO simulations. The RK4-method was noted to be at least as fast as the PISO method.
3. The RK-methods have a low level of numerical dissipation. The provided numerical tests showed that PISO is a relatively non-dissipative algorithm as well.
4. A possible caveat of the current PISO implementation in the OpenFOAM® code is the undocumented flux correction term. In the absence of energy sources this term is of dissipative character. However, in DNS of a channel flow and the lid driven cavity flow, the flux correction term of the PISO-method did not show significant impact on the results.

Acknowledgements

The present study has been funded by the post-doctoral program of the Aalto University. Financial support for research visits from the HPC Europa 2 research program is deeply acknowledged. The computational resources for this study were provided by CSC – Finnish IT Center for Science and SARA, the Netherlands.

References

[1] Canuto C, Hussaini MY, Quarteroni A, Zang T. Spectral methods. Springer; 2007. ISBN 978-3-540-30727-3.
[2] Hirsch C. Numerical computation of internal and external flows. 2nd ed. Elsevier; 2007.
[3] Issa RI. Solution of the implicitly discretized fluid flow equations by operator-splitting. J Comput Phys 1985;62:40–65.
[4] Issa RI, Gosman AD, Watkins AP. Solution of the implicitly discretized fluid flow equations by operator-splitting. J Comput Phys 1986;62:66–82.
[5] Boersma BJ. A 6th order staggered compact finite difference method for the incompressible Navier–Stokes and scalar transport equations. J Comput Phys 2011;230:4940–54.
[6] Boersma BJ. A staggered compact finite difference formulation for the compressible Navier–Stokes equations. J Comput Phys 2005;208:2.
[7] Grinstein F, Fureby C, DeVore C. On MILES based on flux limiting algorithms. Int J Numer Meth Fluids 2005;47:1043–51.
[8] Moser, Kim, Mansour. DNS of turbulent channel flow up to $Re_\tau = 590$. Phys Fluids 1999;11:943–5.
[9] Yang Y, Kaer SK. Large-eddy simulations of the non-reactive flow in the Sydney swirl burner. Int J Heat Fluid Flow 2012;36:47–57.
[10] Kosaka H, Nomura Y, Nagaoka M, Inagaki M, Kubota M. A fractal-based flame propagation model for large eddy simulation. J Engine Res 2011;12:393–401.
[11] Kaario O, Vuorinen V, Hulkkonen T, Keskinen K, Nuutinen M, Larmi M, et al. Large eddy simulation of high gas density effects in fuel sprays. Atom Sprays 2013;23(4):297–325.
[12] Bharadwaj N, Rutland CJ, Chang S. Large eddy simulation modelling of spray-induced turbulence effects. Int J Engine Res 2009;10:121–32.
[13] Rutland CJ. Large-eddy simulations for internal combustion engines – a review. Int J Engine Res 2011;12:421–51.
[14] Wehrfritz A, Vuorinen V, Kaario O, Larmi M. Large eddy simulation of high velocity fuel sprays: studying mesh resolution and breakup model effects for spray A. Atom Sprays 2013;23(5):419–42.
[15] Keskinen J-P, Vuorinen V, Larmi M. Large eddy simulation of flow over a valve in a simplified cylinder geometry. SAE Paper 2011-01-0843; 2011.
[16] Keskinen J-P, Vuorinen V, Kaario O, Larmi M. Large eddy simulation of the intake flow in a realistic single cylinder configuration. SAE Paper 2012-01-0137; 2012.
[17] Vermorel O. Towards the understanding of cyclic variability in a spark ignited engine using multi-cycle LES. Combust Flame 2009;156:15251541.
[18] Goryntsev D, Sadiki A, Klein M, Janicka J. Large eddy simulation based analysis of the effects of cycle-to-cycle variations on air-fuel mixing in realistic DISI IC-engines. Proc Combust Inst 2009;32:2759–66.
[19] Yu R, Bai XS. A semi-implicit scheme for large Eddy simulation of piston engine flow and combustion. Int J Numer Methods Fluids 2012. http://dx.doi.org/10.1002/fld.
[20] Lysenko D, Ertesvg I, Rian K. Modeling of turbulent separated flows using OpenFOAM. Comput Fluids 2012;80(10):1–13.
[21] Flores F, Garreaud R, Munoz RC. CFD simulations of turbulent buoyant atmospheric flows over complex geometry: solver development in OpenFOAM. Comput Fluids 2012;80:408–22.
[22] Weller H. A tensorial approach to computational continuum mechanics using object-oriented techniques. Comput Phys 1998;12:6.
[23] Jasak H, Weller H. Application of the finite volume method and unstructured meshes to linear elasticity. Int J Numer Meth Eng 2000;48:267–87.
[24] Greenshields C, Weller H, Gasparini L, Reese J. Implementation of semi-discrete, non-staggered central schemes in a colocated, polyhedral, finite volume framework for high-speed viscous flows. Int J Numer Meth Fluids 2009;63(1):1–21.
[25] Tukovic Z, Jasak H. A moving mesh finite volume interface tracking method for surface tension dominated interfacial fluid flow. Comput Fluids 2012;55:70–84.
[26] OpenCFD Ltd. User and programmer's guide; 2012. <http://www.openfoam.com>.
[27] Vuorinen V, Hillamo H, Kaario O, Larmi M, Fuchs L. Large eddy simulation of droplet stokes number effects on turbulent spray shape. Atom Sprays 2010;20(2):93–114.
[28] Vuorinen V, Hillamo H, Kaario O, Larmi M, Fuchs L. Large eddy simulation of droplet stokes number effects on mixture quality in fuel sprays. Atom Sprays 2010;20(5):435–51.
[29] Vuorinen V, Larmi M, Sclatter P, Fuchs L, Boersma B. A low-dissipative, scale-selective discretization scheme for the Navier–Stokes equations. Comput Fluids 2012;70(30):195–205.
[30] Vuorinen V, Duwig C, Yu J, Tirunagari S, Kaario O, Larmi M, et al. Large-eddy simulation of highly underexpanded transient gas jets. Phys Fluids 2013;25:016101.
[31] Vuorinen V, Wehrfritz A, Yu J, Kaario O, Larmi M, Boersma B. Large-eddy simulation of subsonic jets. J Phys: Conf Ser 2011;318:032052.
[32] Ni M, Abdou M. A bridge between projection methods and SIMPLE type methods for incompressible Navier–Stokes equations. Int J Numer Meth Eng 2007;72:1490–512.
[33] Chorin AJ. Numerical solution of the Navier–Stokes equations. Math Comput 1968;22:745–62.
[34] Temam R. Une méthode d'approximation des solutions des équations Navier–Stokes. Bull Soc Math France 1968;98:115–52.
[35] Jameson A, Schmidt W, Turkel E. Numerical simulation of the euler equations by finite volume methods using Runge–Kutta time-stepping schemes. In: AIAA 5th computational fluid dynamics conference. AIAA Paper 81-1259; 1981.
[36] Rhie CM, Chow WL. Numerical study of the turbulent flow past an airfoil with trailing edge separation. AIAA J 1983;21(11):1525–32.
[37] Udwadia, Firdaus E, Faragani A. Accelerated Runge–Kutta methods. Discrete Dyn Nat Soc 2008;2008:790619.
[38] Schiesser WE. Computational mathematics in engineering and applied science: ODEs, DAEs and PDEs. CRC Press; 1994. ISBN 0-8493-7373-5.
[39] Duwig C, Salewski M, Fuchs L. Large eddy simulation of a turbulent flow past two symmetric backward-facing steps: a sensitivity analysis. AIAA J 2008;46(2):408–19.
[40] Jasak H, Jemcov A, Maruszewski JP. Preconditioned linear solvers for large eddy simulation. In: 15th Annual conference of the CFD society of Canada, May 27–31, 2007. CFD Society of Canada; 2007.
[41] Shashank, Larson J, Iaccarino G. A co-located incompressible Navier–Stokes solver with exact mass, momentum and kinetic energy conservation in the inviscid limit. J Comput Phys 2010;229:4425–30.
[42] Erturk E, Corke T, Gökcöl. Numerical solutions of 2-D steady incompressible driven cavity flow at high Reynolds numbers. Int J Numer Meth Fluids 2007;72:1490–512.
[43] Sengupta TK, Dipankar A, Sagaut P. Error dynamics: beyond von Neumann analysis. J Comput Phys 2007;226:1211–8.
[44] Rajpoot MK, Sengupta TK, Dutt PK. Optimal time advancing dispersion relation preserving schemes. J Comput Phys 2010;229:3623–51.